ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and we are privileged to have got this all along the completion of our project. All that we have done is only due to such supervision and assistance and we would not forget to thank them.

We owe our deep gratitude to our Computer Science teacher Ms. Simrat Kaur, for believing in our capabilities and guiding us all along till the completion of our project work by providing all the necessary information for developing a good system. We take this opportunity to express our deep sense of gratitude for her invaluable guidance and constant encouragement, immense motivation, which has sustained our efforts at all stages of this project work.

Certificate

This is to certify that the project entitled

SKYLON

is the work of **SUJAL SINGH** in class **XII-D**

Roll No. **25**, school **St. Xavier's School, Rohini** during the academic year 21/22, is prepared under the supervision and guidance of

MS. SIMRAT KAUR

Teacher's Signature **Student's Signature**

INDEX

components that sit on top of your browser covering less than 4-5% of the available screen area. Although the chrome is not limited to just the GUI for navbar, it’s also responsible for things like user data collection, basically anything that is not the second component comes under the chrome. The other part where the actual web pages are displayed is created by the second component called the **rendering engine**. The later component is the one that is difficult to develop.

The chrome of our browser has been developed with **PyQT5**, a cross platform GUI framework. The engine is a little more complicated, building an entire engine would be impossible, here’s a proof for this statement:

*Firefox has 21 million lines of code, if on an average each line has 20 words and the programmer was to type non-stop 24/7 at a constant speed of 50 words per minute, it would take 16 years to make a browser like Firefox: (21,000,000 * 20) / (50 * 60 * 24 * 365) = ~16 years*

Realizing this problem, we decided to create a new engine to accomplish our goals but still maintain backwards compatibility by using the **chromium embedded framework.**

We provide the user with the choice to select whether they would like to use embedded chromium or our new engine. **CEF Python** brings CEF to python and so it was required to use the chromium embedded framework with python.

If its not obvious what tokenizing does, here’s an example:

If someone was to develop a tokenizer for the English language the sentence “the cat sits” would be converted into tokens like {“the”: “article”, “cat”: “noun”, “sits”: “verb”}.

The next stage would further take these tokens to divide it into subject and predicate.

SYSTEM REQUIREMENTS

Hardware:

Component	Minimum	Recommended
CPU	*Dual core x86/x64 (4 threads)*	*Quad core x64 (8 threads)*
GPU	*Intel UHD / AMD Vega*	*GTX 1650 / Radeon RX 570*
RAM	*4 GB DDR4*	*8 GB DDR4*
Storage	*~ 200 MB*	*~ 300 MB (preferably SSD)*

Software:

- **Python** 3 - 3.8
  (Note: Because CEF Python currently lacks support for python 3.8+, higher versions will not work)
- **GNU/Linux**, all major distributions are supported -
  (Fedora, Debian and its derivatives, RedHat, CentOS, Gentoo, Arch, OpenSUSE, Manjaro, ...)

  or

  Windows (minor changes are required to run on windows)

  or

  MacOS (minor changes are required to run on MacOS)

Tested on AMD Ryzen 5 3600 running Fedora Linux with GTX 1650 4GB graphics card, 8 GB DDR4 3200MHz RAM, 256 GB SSD

PYTHON

(AS FRONTEND)

Python offers a significant advantage over other languages for our project. As explained above, our project has to parse HTML and CSS and so a lot of string functionality is required, which happens to be built right into python's easy syntax and also a large set of in-built functions. Python is object oriented and supports object inheritance allowing us to easily create nodes for the DOM tree in the parser.

Python's clear, easy and feature rich syntax attracts more developers to the language, consequently python's community is quite large and so if one ever runs into any problems it's likely someone else has faced a similar issue and a solution exists for it. Additionally, huge projects like CEF Python (which is being used in our project) are ported to python too.

Some other reasons to use python: it's open source! And so free of cost. Anyone can modify the language to one's needs. As a matter of fact, many have done these modifications, implementations like PyPy allow one to eliminate one of python's major drawbacks that is speed, giving our project scope for further speed enhancements. Also, Python is cross-platform, so you write the same code (in most cases) for all platforms. Python also has large availability for libraries, so one doesn't need to reinvent the wheel every time. This allows for quicker development compared to other languages. Even ignoring the vast library availability, development is quicker in python as one doesn't have to declare variable types.

In conclusion, the choice to use python as the front-end was because it has a simple syntax, large community, it's open source and cross-platform, it has endless applications, large availability of libraries and offers small development times.

MySQL

(AS BACKEND)

MySQL is a Relational Data Base Management System **(RDBMS)**. A relational database organizes data into one or more data tables in which data types may be related to each other; these relations help structure the data. MySQL uses Structured Query Language **(SQL)** as used by most other such databases. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

MySQL is also open source, and so it is free. It is also cross platform therefore can be deployed on a server with any major operating system. Instead of locally deploying our MySQL server, we have deployed our MySQL server on AWS cloud running Ubuntu distribution of GNU/Linux.

Why binary files were also used?

Instead of making a request to the MySQL server every time user data is required, the browser caches user data with a binary file to allow for quicker access.

DEPENDECIES

- **PyQT5**
  This is a cross platform GUI framework. To create a window, complicated calls have to be made to the operating system and the specific calls vary with each operating system. To avoid the hassle of unnecessarily dealing with these already solved problems, a GUI framework has been used.

- **CEF Python**
  Chromium Embedded Framework allows embedding chromium browser inside a window. CEF Python is the python version for CEF. As mentioned before this is being used to maintain backwards compatibility for websites written with JS. Also, since the Skylon engine doesn’t yet support variety of node types and layouts, welcome launcher of our browser is utilizing CEF Python to create a beautiful GUI. (CEF Python can create a standalone window without being embedded into another window)

USER DEFINED FUNCTIONS (UDF)

Name	Parameters	Return Type	Description
is_first_launch	*None*	*Boolean*	*Returns True if binary file 'user.cache' does not exist else returns False*
get_user_data	*None*	*List*	*Reads 'user.cache' and returns data, defaults to ["John", "Doe", "johndoe@example.com", "chromium"]*
launch	*None*	*None*	*Launches WelcomeLauncher if first launch else launches the browser of choice.*
WelcomeLauncher	*None*	*None*	*Launches Chromium with page as https://sujalsinghx86.github.io/school-project-web/ and exposes python functions to the web page.*
launch_chromium	*None*	*None*	*Launches browser with chromium as engine choice*
launch_skylon	*None*	*None*	*Launches browser with Skylon as browser choice.*
create_user	*first_name, last_name, email, engine_choice*	*None*	*Creates a new user in the MySQL Database*
user_exists	*first_name, last_name, email*	*Boolean*	*Checks if a user with given parameters exists or not by querying MySQL database.*
update_preference	*email, new_preference*	*None*	*Updates engine preference for given email parameter on MySQL database with new_preference parameter*
delete_user	*email*	*None*	*Deletes user entry from MySQL database for given email parameter*
cache_user	*email*	*None*	*Fetches user data from MySQL database for given email parameter and creates binary file 'user.cache' for quicker access.*

GLOSSARY

Browser Engine: This is the component of the browser responsible for all the core functionalities of a browser such as networking, parsing, rendering and display.

Chrome: All the functionality except the browser's engine is termed as the chrome of the browser.

DOM: Document Object Model is an HTML Tree consisting of HTML elements as nodes which can be used as an API to manipulate the tree after a page is loaded.

Parsing: (Here) The process of tokenizing and constructing tree either for HTML or CSS to construct a DOM

Tokenizing: The process of breaking down a string into a list of lexical components as defined by a grammar for a specific language (here HTML and CSS) through a series of steps. These steps depend upon type of grammar of the language.

Tree construction: The process of converting a list of tokens into a tree structure through a series of steps, in our project these steps conform to the HTML and CSS specification.

FUTURE ASPECTS

This project opens a lot of doors for future development.

One primary aspect for development will be the Skylon engine itself which can be developed further to support a variety of features supported by the modern web such as SVG & raster image formats, XML, etc. The engine will also need a lot of development in the rendering module. The engine currently only supports text elements, further development would allow other nodes and much more layout options. Another thing which should be implemented in the future is a DOM API in python to be accessible by web pages rendered by the Skylon engine, just like it's current accessible in JS for pages rendered by modern browser engines.

Another thing which allows further development is shifting the entire code base to the PyPy implementation of python, this would allow significant speed gains.

Additionally, the chrome of our browser could support multiple tabs, page history, bookmarks, etc in the future.

All in all, this project has taught us a lot and we plan to further develop this during our post-secondary education. One day we might stand a chance at competing against all the major browsers in existence today.

BIBLIOGRAPHY

HTML SPECIFICATIONS

https://html.spec.whatwg.org/multipage/parsing.html

https://html.spec.whatwg.org/#rendering

CSS SPECIFICATIONS

https://www.w3.org/TR/css/

https://www.w3.org/TR/css-syntax-3/

https://www.w3.org/TR/css-box-3/

CEFPYTHON

https://github.com/cztomczak/cefpython

EBOOKS

https://browser.engineering/

https://htmlparser.info/